**集合框架**

**Java集合框架由Java类库的一系列接口、抽象类以及具体实现类组成。**我们这里所说的集合就是把一组对象组织到一起，然后再根据不同的需求操纵这些数据。集合类型就是容纳这些对象的一个容器。也就是说，最基本的集合特性就是把一组对象放一起集中管理。根据集合中是否允许有重复的对象、对象组织在一起是否按某种顺序等标准来划分的话，集合类型又可以细分为许多种不同的子类型。

**Java集合框架为我们提供了一组基本机制以及这些机制的参考实现，其中基本的集合接口是Collection接口，其他相关的接口还有Iterator接口、RandomAccess接口等。**这些集合框架中的接口定义了一个集合类型应该实现的基本机制，Java类库为我们提供了一些具体集合类型的参考实现，根据对数据组织及使用的不同需求，只需要实现不同的接口即可。Java类库还为我们提供了一些抽象类，提供了集合类型功能的部分实现，我们也可以在这个基础上去进一步实现自己的集合类型。

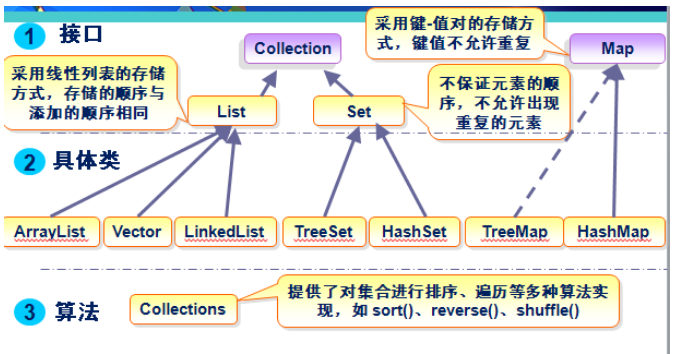
## 一、集合与数组

    1、数组固定长度；集合长度可变。

    2、数组可以存储基本数据类型，也可以存储引用数据类型（即对象）；集合只能存储引用数据类型（对象）；

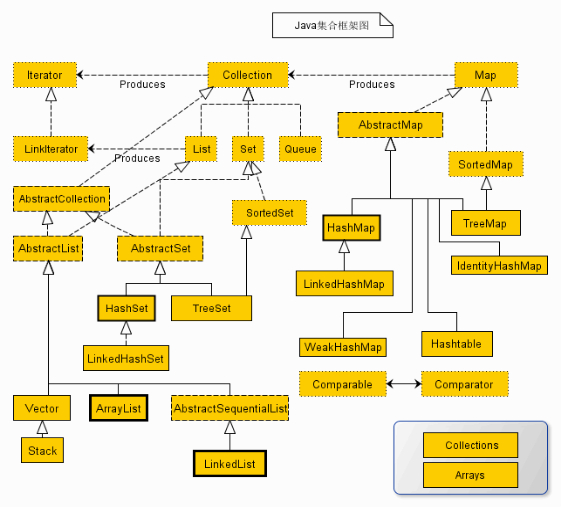
    3、数组存储的元素必须是同一个数据类型；集合存储的对象可以是不同的数据类型。

4、数组不适合在数量未知的情况下使用；而集合在多数情况下都能使用。



## 二、层次关系

如图所示：图中，实线边框的是实现类，折线边框的是抽象类，而点线边框的是接口



**1 Collection接口是集合类的根接口，Java中没有提供这个接口的直接的实现类。**但是却让其被继承产生了两个接口，就是Set和List。Set中不能包含重复的元素。List是一个有序的集合，可以包含重复的元素，提供了按索引访问的方式。

Map是Java.util包中的另一个接口，它和Collection接口没有关系，是相互独立的，但是都属于集合类的一部分。Map包含了key-value对。Map不能包含重复的key，但是可以包含相同的value。

Collection接口是集合层级结构的根接口。一个集合代表了一组对象，这组对象被称为集合的元素。一些集合允许重复的元素而其他不允许；一些是有序的而一些是无序的。Java类库中并未提供任何对这个接口的直接实现，而是提供了对于它的更具体的子接口的实现（比如Set接口和List接口）。



**2 Iterator，所有的集合类，都实现了Iterator接口，**这是一个用于遍历集合中元素的接口，主要包含以下三种方法：  
1.hasNext()是否还有下一个元素。  
2.next()返回下一个元素。  
3.remove()删除当前元素。

迭代器就是一个我们用来遍历集合中的对象的东西。也就是说，对于集合，我们不是像对原始类型数组那样通过直接访问元素来迭代，而是通过迭代器来遍历对象。这么做的好处是将对于集合类型的遍历行为与被遍历的集合对象分离，这样一来我们无需关心该集合类型的具体实现是怎样的。只要获取这个集合对象的迭代器, 便可以遍历这个集合中的对象了。而像遍历对象的顺序这些细节，全部由它的迭代器来处理。现在我们来梳理一下前面提到的这些东西：首先，Collection接口实现了Iterable<E>接口，这意味着所有实现了Collection接口的具体集合类都是可迭代的。那么既然要迭代，我们就需要一个迭代器来遍历相应集合中的对象，所以Iterable<E>接口要求我们实现iterator方法，这个方法要返回一个迭代器对象。一个迭代器对象也就是实现了Iterator<E>接口的对象，这个接口要求我们实现hasNext()、next()、remove()这三个方法。其中hasNext方法判断是否还有下一个元素（即是否遍历完对象了），next方法会返回下一个元素（若没有下一个元素了调用它会引起抛出一个NoSuchElementException异常），remove方法用于移除最近一次调用next方法返回的元素（若没有调用next方法而直接调用remove方法会报错）。我们可以想象在开始对集合进行迭代前，有个指针指向集合第一个元素的前面，第一次调用next方法后，这个指针会”扫过"第一个元素并返回它，调用hasNext方法就是看这个指针后面还有没有元素了。也就是说这个指针始终指向刚遍历过的元素和下一个待遍历的元素之间。

## 三、几种重要的接口和类简介

**1、List（有序、可重复）**  
List里存放的对象是有序的，同时也是可以重复的，List关注的是索引，拥有一系列和索引相关的方法，查询速度快。因为往list集合里插入或删除数据时，会伴随着后面数据的移动，所有插入删除数据速度慢。



1. **ArrayList类特点**

1）元素可以重复；

2）非同步的集合类；

3）实现了ICollection和List接口；

4）容量可以动态增加和减少，大小可变，初始容联为10；

5）插入、删除效率低下，查询效率高。

ArrayList<E>是一个可动态调整大小的数组，允许null类型的元素。我们知道，Java中的数组大小在初始化时就必须确定下来，而且一旦确定就不能改变，这会使得在很多场景下不够灵活。ArrayList<E>很好地帮我们解决了这个问题，当我们需要一个能根据包含元素的多少来动态收缩伸张的数组时，那么ArrayList<E>正是我们所需要的。

们先来看看这个类的常用方法：

boolean add(E e) //添加一个元素到列表末尾

void add(int index, E element) //添加一个元素到列表的指定位置

void clear()

boolean contains(Object o)

void ensureCapacity(int minCapacity) //确保ArrayList至少能容纳参数指定数目的对象，若有需要会增加ArrayList实例的容量。

E get(int index) //返回指定位置的元素

int indexOf(Object o)

boolean isEmpty()

Iterator<E> iterator()

ListIterator<E> listIterator()

E remove(int index)

boolean remove(Object o)

E set(int index, E element)

int size()

当我们插入了比较多的元素，导致ArrayList快要装满时，它会自动增长容量。而自动增长容量是通过创建一个新的容量更大的ArrayList对象，再把原来的所有元素复制过去实现的。若要想避免这种开销，在知道大概会容纳多少数据时，我们可以在构造时指定好它的大小以尽量避免它自动增长的发生；我们也可以调用ensureCapacity方法来增加ArrayList对象的容量到我们指定的大小。

ArrayList有以下三个构造器：

ArrayList()

ArrayList(Collection<? extends E> c)

ArrayList(int initialCapacity)

1. **LinkList类特点**

1）插入、删除元素比较快，查询效率低；

2）元素可以为null；

3）双向链表、堆栈、队列；

4）非同步的集合类。

1. **Set（无序、不能重复）**  
   Set里存放的对象是无序，不能重复的，集合中的对象不按特定的方式排序，只是简单地把对象加入集合中。
2. **Map（键值对、键唯一、值不唯一）**  
   Map集合中存储的是键值对，键不能重复，值可以重复。根据键得到值，对map集合遍历时先得到键的set集合，对set集合进行遍历，得到相应的值。

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **是否有序** | **是否允许元素重复** |
| **Collection** | |  |  |
| **List** | | 是 | 是 |
| **Set** | **AbstractSet** | 否 | 否 |
|  | **HashSet** |
|  | **TreeSet** | 是（用二叉排序树） |
| **Map** | **AbstractMap** | 否 | 使用key-value来映射和存储数据，key必须唯一，value可以重复 |
|  | **HashMap** |
|  | **TreeMap** | 是（用二叉排序树） |

## 四、遍历

 在类集中提供了以下四种的常见输出方式：

1. Iterator：迭代输出，是使用最多的输出方式。

2）ListIterator：是Iterator的子接口，专门用于输出List中的内容。

3）foreach输出：JDK1.5之后提供的新功能，可以输出数组或集合。

4）for循环

 for的形式：for（int i=0;i<arr.size();i++）{...}

 foreach的形式： for（int　i：arr）{...}

 iterator的形式：  
Iterator it = arr.iterator();  
while(it.hasNext()){ object o =it.next(); ...}

## 五、ArrayList和LinkedList

ArrayList和LinkedList在用法上没有区别，但是在功能上还是有区别的。LinkedList经常用在增删操作较多而查询操作很少的情况下，ArrayList则相反。

因为ArrayList是使用数组实现的,若要从数组中删除或插入某一个对象，需要移动后段的数组元素，从而会重新调整索引顺序,调整索引顺序会消耗一定的时间，所以速度上就会比LinkedList要慢许多. 相反,LinkedList是使用链表实现的,若要从链表中删除或插入某一个对象,只需要改变前后对象的引用即可!

## 六、Map集合

**实现类：HashMap、Hashtable、LinkedHashMap和TreeMap**

**1 HashMap**

HashMap是最常用的Map，它根据键的HashCode值存储数据，根据键可以直接获取它的值，具有很快的访问速度，遍历时，取得数据的顺序是完全随机的。因为键对象不可以重复，所以HashMap最多只允许一条记录的键为Null，允许多条记录的值为Null，是非同步的

**2 Hashtable**

Hashtable与HashMap类似，是HashMap的线程安全版，它支持线程的同步，即任一时刻只有一个线程能写Hashtable，因此也导致了Hashtale在写入时会比较慢，它继承自Dictionary类，不同的是它不允许记录的键或者值为null，同时效率较低。

**3 ConcurrentHashMap**

线程安全，并且锁分离。ConcurrentHashMap内部使用段(Segment)来表示这些不同的部分，每个段其实就是一个小的hash table，它们有自己的锁。只要多个修改操作发生在不同的段上，它们就可以并发进行。

**4 LinkedHashMap**

LinkedHashMap保存了记录的插入顺序，在用Iteraor遍历LinkedHashMap时，先得到的记录肯定是先插入的，在遍历的时候会比HashMap慢，有HashMap的全部特性。

**5 TreeMap**

TreeMap实现SortMap接口，能够把它保存的记录根据键排序，默认是按键值的升序排序（自然顺序），也可以指定排序的比较器，当用Iterator遍历TreeMap时，得到的记录是排过序的。不允许key值为空，非同步的；

**6 map的遍历**

**第一种：KeySet()**  
将Map中所有的键存入到set集合中。因为set具备迭代器。所有可以迭代方式取出所有的键，再根据get方法。获取每一个键对应的值。 keySet():迭代后只能通过get()取key 。  
取到的结果会乱序，是因为取得数据行主键的时候，使用了HashMap.keySet()方法，而这个方法返回的Set结果，里面的数据是乱序排放的。  
典型用法如下：  
Map map = new HashMap();  
map.put("key1","lisi1");  
map.put("key2","lisi2");  
map.put("key3","lisi3");  
map.put("key4","lisi4");    
//先获取map集合的所有键的set集合，keyset（）  
Iterator it = map.keySet().iterator();  
 //获取迭代器  
while(it.hasNext()){  
Object key = it.next();  
System.out.println(map.get(key));  
}

**第二种：entrySet（）**  
Set<Map.Entry<K,V>> entrySet() //返回此映射中包含的映射关系的 Set 视图。（一个关系就是一个键-值对），就是把(key-value)作为一个整体一对一对地存放到Set集合当中的。Map.Entry表示映射关系。entrySet()：迭代后可以e.getKey()，e.getValue()两种方法来取key和value。返回的是Entry接口。典型用法如下：  
Map map = new HashMap();  
map.put("key1","lisi1");  
map.put("key2","lisi2");  
map.put("key3","lisi3");  
map.put("key4","lisi4");  
//将map集合中的映射关系取出，存入到set集合  
Iterator it = map.entrySet().iterator();  
while(it.hasNext()){  
Entry e =(Entry) it.next();  
System.out.println("键"+e.getKey () + "的值为" + e.getValue());  
}  
推荐使用第二种方式，即entrySet()方法，效率较高。  
对于keySet其实是遍历了2次，一次是转为iterator，一次就是从HashMap中取出key所对于的value。而entryset只是遍历了第一次，它把key和value都放到了entry中，所以快了。两种遍历的遍历时间相差还是很明显的。

## 七、主要实现类区别小结

**1 Vector和ArrayList**  
（1）vector是线程同步的，所以它也是线程安全的，而arraylist是线程异步的，是不安全的。如果不考虑到线程的安全因素，一般用arraylist效率比较高。  
（2）如果集合中的元素的数目大于目前集合数组的长度时，vector增长率为目前数组长度的100%，而arraylist增长率为目前数组长度的50%。如果在集合中使用数据量比较大的数据，用vector有一定的优势。  
（3）如果查找一个指定位置的数据，vector和arraylist使用的时间是相同的，如果频繁的访问数据，这个时候使用vector和arraylist都可以。而如果移动一个指定位置会导致后面的元素都发生移动，这个时候就应该考虑到使用linklist,因为它移动一个指定位置的数据时其它元素不移动。  
ArrayList 和Vector是采用数组方式存储数据，此数组元素数大于实际存储的数据以便增加和插入元素，都允许直接序号索引元素，但是插入数据要涉及到数组元素移动等内存操作，所以索引数据快，插入数据慢，Vector由于使用了synchronized方法（线程安全）所以性能上比ArrayList要差，LinkedList使用双向链表实现存储，按序号索引数据需要进行向前或向后遍历，但是插入数据时只需要记录本项的前后项即可，所以插入数度较快。

**2 arraylist和linkedlist**  
（1）ArrayList是实现了基于动态数组的数据结构，LinkedList基于链表的数据结构。  
（2）对于随机访问get和set，ArrayList觉得优于LinkedList，因为LinkedList要移动指针。  
（3）对于新增和删除操作add和remove，LinkedList比较占优势，因为ArrayList要移动数据。 这一点要看实际情况的。若只对单条数据插入或删除，ArrayList的速度反而优于LinkedList。但若是批量随机的插入删除数据，LinkedList的速度大大优于ArrayList. 因为ArrayList每插入一条数据，要移动插入点及之后的所有数据。

**3 HashMap与TreeMap**  
（1） HashMap通过hashcode对其内容进行快速查找，而TreeMap中所有的元素都保持着某种固定的顺序，如果你需要得到一个有序的结果你就应该使用TreeMap（HashMap中元素的排列顺序是不固定的）。  
（2）在Map 中插入、删除和定位元素，HashMap是最好的选择。但如果您要按自然顺序或自定义顺序遍历键，那么TreeMap会更好。使用HashMap要求添加的键类明确定义了hashCode()和 equals()的实现。  
两个map中的元素一样，但顺序不一样，导致hashCode()不一样。同样做测试：  
**在HashMap中，同样的值的map,顺序不同，equals时，false;  
而在treeMap中，同样的值的map,顺序不同,equals时，true，说明，treeMap在equals()时是整理了顺序了的。**

**4 HashTable与HashMap**  
（1）同步性:Hashtable是线程安全的，也就是说是同步的，而HashMap是线程序不安全的，不是同步的。  
（2）HashMap允许存在一个为null的key，多个为null的value 。

（3）hashtable的key和value都不允许为null。